1. Introduction

This document provides a complete description of the HTTP Distribution and Replication Protocol (DRP). The goal of the DRP protocol is to significantly improve the efficiency and reliability of data distribution over HTTP. Here is a more detailed list of goals:

- Provide a widely applicable mechanism for the efficient replication of data and content.
- Improve the efficiency and reliability of caching servers and proxies.
- Significantly improve the efficiency of subscription-based data and content distribution.
- Improved reliability and versioning for the distribution of mission-critical applications and data.
- Create an interoperability standard for replicating content between clients and servers from different vendors.
- Keep the protocol simple to avoid incompatibilities.
- Provide functionality that is complementary to existing standards.
- Provide functionality that is backward compatible with existing HTTP servers and proxies.
- Provide functionality that can be deployed anywhere where HTTP is available today.

The HTTP Distribution and Replication Protocol was designed to efficiently replicate a hierarchical set of files to a large number of clients. No assumption is made about the content or type of the files; they are simply files in some hierarchical organization.

After the initial download a client can keep the data up-to-date using the DRP protocol. Using DRP the client can download only the data that has changed since the last time it checked. Downloading only the differences is much more efficient because data typically evolves slowly over time, and because changes are usually restricted to only a subset of the data. One of the goals of the DRP protocol is to avoid downloading the same data more than once.

The data that is distributed can consist of more than just HTML pages; it can consist of any kind of code or content. The DRP protocol provides strong guarantees about data versioning. This is a requirement for distributing mission critical applications, because getting the correct version of each component is a necessary to ensure a complete application will work correctly. Correct file versioning using existing HTTP requests is problematic because there is currently no reliable mechanism for identifying a particular version of a file.

The DRP protocol uses content identifiers to automatically share resources that are requested more than once. This eliminates redundant transfers of commonly used resources. The content identifiers used in the DRP protocol are based on widely accepted checksum technology.

The DRP protocol uses a data structure called an *index*, which is currently specified using the eXtensible Markup Language (XML). Because the index describes meta data, we anticipate using the Resource Description Framework (RDF), in a future version of the DRP protocol specification. XML is used in the interim because the RDF standard was not finalized at the time of writing.

The DRP protocol relies on existing HTTP/1.1 functionality to achieve better performance when replicating files, and it is backward compatible with existing HTTP/1.0 and HTTP/1.1 proxies so that it can be deployed immediately. To further improve the download efficiency and caching behavior of HTTP, the proposal introduces new HTTP header information that may become part of the core HTTP protocol specification in the future. Because the DRP protocol is layered on HTTP, it will benefit from ongoing efforts in the HTTP-NG working group.

The DRP protocol is based on technology which was originally developed, implemented, and deployed by Marimba Inc. Although it was originally designed for software distribution, it is widely applicable in many different areas, and it enables the large scale automatic distribution of software, data, and content to many different clients.

2. The HTTP Distribution and Replication Protocol

2.1 Content Identifiers

The DRP protocol uses content identifiers to identify individual pieces of content. A content identifier is a token that can be used to uniquely identify any piece of data or content. It can also be used to determine whether two pieces of content are identical with great accuracy.

A content identifier consists of one or more Uniform Resource Identifiers (URI) separated by commas. The URIs are combined together into a single content identifier and form a globally unique identifier. Here is the syntax of a content-identifier:

```
content-identifier = URI ( "," URI )*
```

Typically a checksum algorithm is used to generate a content identifier for a piece of content. One of the checksum algorithms which can be used is the Message Digest algorithm from RSA. The MD5 algorithm is a well-known algorithm for computing a 128-bit checksum for any file or object. See http://www.rsa.com/pub/rfc1321.txt for details on the implementation of the MD5 algorithm.

The likelihood of two different files producing the same MD5 checksum is very small (about 1 in 2^64), and as such, the MD5 checksum of a file can be used to construct a reliable content identifier that is very likely to uniquely identify the file. The reverse is also true. If two files have the same MD5 checksum, it is very likely that the files are identical.

It is possible to define a Universal Resource Name (URN), which is a kind of URI, for a 128-bit MD5 checksum:

```
MD5-URN = "urn:md5:" base64-number
```

Another checksum algorithm used in the DRP protocol is the SHA algorithm from the National Institute of Standards and Technology. It is similar to the MD5 algorithm but its checksums are 160 bits long, and have different properties. See http://csrc.nist.gov/fips/fip180-1.txt for details on the implementation of the SHA algorithm.

Here is how a URN is created using a 160-bit SHA checksum:

```
SHA-URN = "urn:sha:" base64-number
```

Both MD5 and SHA based URIs use base64 encoding to encode the checksum of the content. The base64 algorithm encodes each 3 bytes of the checksum in 4 characters containing 6 bits of information

each. The details of the base64 encoding algorithm are part of the MIME specification.

Note that a base64 number can include '/' and '+' characters. Although these characters are treated specially in the URN syntax specification, it is not necessary to escape them when used in a checksum URN.

If a content identifier contains a URI that refers to a well known checksum algorithm such as MD5 or SHA, it is possible to verify the integrity of the content. If none of the URIs in the content identifier refer to a known checksum algorithm, the content identifier should be treated as an opaque string that can be used for addressing purposes only.

Here are some examples of valid content identifiers:

```
urn:md5:FNG4c6MJLdDEY1rcoGb4pQ==

urn:md5:HUXZLQLMuI/KZ5KDcJPcOA==

urn:sha:thvDyvhfIqlvFe+A9MYgxAfm1q5=

http://www.acme.com/images/foo.gif,urn:md5:FNG4c6MJLdDEY1rcoGb4pQ==

http://www.acme.com/Example/,urn:x-version:5

ftp://www.acme.com/Hello%20World
```

In applications where the possibility of duplicates for a given URN is unacceptable, content identifiers can be generated with the required uniqueness by using multiple appropriate URIs. If for example version numbers are used, it is important to further qualify the content identifier in order to make it globally unique. This can be achieved by including the URL of the object to which the version number applies in the content identifier.

A content identifier is often embedded in an HTTP header, and therefore the URIs in the content identifier are not allowed to contain reserved characters such as spaces or comma characters. All reserved characters should be encoded as specified in the URI specification.

2.2 Index Format

To describe the exact state of a set of data files, the DRP protocol uses a data-structure called an *index*. An index not only describes the hierarchical structure of the files, but it also describes the version, size, and type of each file. An index is a snapshot of the state of a set of files at a particular moment in time.

An index is typically stored in memory as a tree data structure, but in order for clients and servers to communicate this information over HTTP, an index can be described using XML. The index DTD used by the DRP protocol is specified in Appendix A. Using this DTD, here is an example of an index that describes a hierarchical set of files:

```
<?XML VERSION="1.0" RMD="NONE"?>
<index>
<file path="home.html"    size="12345"  id="urn:md5:PBPjWBDv/sd9alS9BYuX0w=="/>
<file path="layer1.js"    size="32112"  id="urn:md5:W25YCu3toJt3ZsDsHIZmpg=="/>
<dir  path="images">
<file path="acme.gif"     size="4532"   id="urn:md5:+hbZN5XfU6QAJB1RF1/KSQ=="/>
<file path="banner.gif"   size="10452"  id="urn:md5:tr3X+oN3r9kqvsiyDSSjjg=="/>
</dir>
<dir  path="java/classes">
<file path="Scroll.java"  size="14323"  id="urn:md5:xjBkgWouS6p6FTUMIkx/Zg=="/>
<file path="gui.jar"      size="540321" id="urn:md5:tcUzw0DKut3SiTpmpAsi8g=="/>
</dir>
</index>
```

2.3 Index Retrieval

A DRP index is retrieved given a URL to the index. The index can be stored in any file and can be retrieved using a normal HTTP GET request. The mime type of the index file allows clients to treat the index as a special file which provides meta information about other files. The client can use the index to automatically download the files that are specified.

The index file can be an ordinary file on an HTTP server, but it can also be generated dynamically. Note that the index isn't necessarily generated from a file system, it could also represent hierarchical data from a different source such as a database.

Once the initial download is complete, a client can update the content by downloading a new version of the index, and comparing it against the previous version of the index. Because each file entry in the index has a content identifier, the client can determine which files have changed and so determine the minimal set of files that need to be downloaded in order to bring the client up-to-date.

Index Base URL

The index file is typically located in the root directory of the file hierarchy it describes. In that case the base URL of the files in the index is the same as the base URL of the index itself. The index file should not be listed in the index itself.

An index can also explicitly specify a different absolute or relative base URL using the **base** attribute of the **index** tag. This allows an index to describe files that are located in a different directory, or even on a different server.

For example, if the index is loaded from the following URL:

```
http://www.acme.com/Examples/index.xml
```

Unless a different base URL is specified in the index tag, the base URL of the index will be:

```
http://www.acme.com/Examples/
```

That means that the index in the example above describes the following files:

```
http://www.acme.com/Examples/home.html
http://www.acme.com/Examples/layer1.js
http://www.acme.com/Examples/images/acme.gif
http://www.acme.com/Examples/images/banner.gif
http://www.acme.com/Examples/java/classes/Scroll.java
http://www.acme.com/Examples/java/classes/gui.jar
```

Index Expiration

In the absence of any meta data, the client should observe the *Expires* field in the HTTP reply of the index request to determine how long the index is valid. When the index expires, a new version of the index should be downloaded.

This provides a simple mechanism for scheduling updates of indexes. In some cases additional meta data will be available which can provide more detailed information for scheduling updates of the index and the content it describes.

Index Caching

An HTTP proxy may cache indexes if the HTTP header indicates that this is allowed. An HTTP proxy should treat indexes just like normal files. The HTTP/1.0 and HTTP/1.1 specification provide a variety of cache control mechanisms that can be used to control the caching of indexes.

A server which generates personalized indexes that all have the same base URL will have to mark the index as not cacheable using the standard HTTP mechanisms as defined in HTTP/1.0 and HTTP/1.1 protocol specifications.

2.4 Content Based Addressing

By requesting an index file it is possible for a client to obtain a complete description of the structure and state of a set of data files. Given an index, it is possible for a client to determine exactly which files need to be downloaded, as well as the total size of the download.

Because the client can determine the exact set of files that need to be downloaded, it is possible to issue

multi-file get requests to get all the files at once. Various HTTP extensions proposed by the HTTP-NG working group, and as part of the HTTP Pipelining proposal, can be used to make the download of multiple files more efficient.

Note that a client can use a disk cache for data files, which is accessed using content identifiers. This means that if multiple indexes refer to a file with the same content identifier, the client can automatically detect that the file is already in the cache, and thus avoid downloading the file a second time. This is not uncommon because different sites often refer to the same standard libraries or images, and because their content identifiers match, multiple redundant downloads can be avoided. Eliminating redundant downloads can reduce the overhead of downloading commonly used data and software components. Using content identifiers it is possible to detect which files are identical, even if the files are from different hosts.

An index is not required to contain a content identifier for each file, and the content identifier can be omitted for any or all of the files in the index. A server can decide to omit the content identifier for files that are dynamically generated, such as a live video picture. If no content identifier is present the file should be retrieved using a normal get-if-modified request. When comparing indexes, files without a content identifier should always be considered to be different.

Content-ID Header Field

Now that it is possible to obtain an index for a large set of files, a mechanism is needed to obtain the correct version of each of the files that need downloading. Note that the correct version of each file is determined by its content identifier as specified in the index. It is very important for the distribution of applications that the correct version of each file is obtained, rather than just the current version.

When requesting a file, the client can include the content identifier in the HTTP GET request to the server. For example:

```
GET /Example/home.html HTTP/1.1
Content-ID: urn:md5:PEFjWBDv/sd9alS9BYuX0w==
```

A new HTTP header field called Content-ID is used to specify the correct version of the file that is requested. The server can use the content-identifier in the Content-ID field server to determine if the requested version of the file can be delivered to the client.

The content identifier of the returned file should be included in the HTTP reply header using the Content-ID header field. The content identifier in the reply must match the requested content identifier. If the correct version of the file is not available, the server should respond with the error: "404 File Version Not Found".

If no content identifier is specified in the HTTP GET request, then the server should return the current version of the file, as is the case in a normal HTTP request. However, the reply should always include the Content-ID field if the correct content identifier is known for the file that is returned.

A server that is unaware of the Content-ID field will always reply with the current version of the requested file, regardless of the content identifier specified in the request. Note that the server is not required to specify a Content-ID in the reply, and that it is the responsibility of the client to verify that the reply contains the correct content identifier if a Content-ID field is present in the reply. When the requested content identifier contains a verifiable checksum URN, the client should always recompute the checksum to verify that the correct content was returned.

2.5 Differential Downloads

When a file is updated on the server, it will be downloaded by each of the clients that needs the new version. Updates to files very often affect only small portions of the file, and it would be much more efficient if the server could reply with only the parts of the file that have changed. This can be achieved using a differential GET request.

Differential-ID Header Field

A client can use the index obtained from a server to determine what changes have occurred in a set of files. It is also possible to detect which files were modified, rather than created or deleted.

When a file is modified the client can issue a *differential* GET request for the file, which includes not only the content identifier of the desired version of the file, but also the content identifier of the current version of the file on the client.

In a differential GET request the content identifier of the file that the client currently possesses is specified using the Differential-ID header field. For example:

```
GET /Example/home.html HTTP/1.1
Content-ID: urn:md5:PEFjWBDv/sd9alS9BYuX0w==
Differential-ID: urn:md5:3FS2oCnWPZptpN05oKBemA==
...
```

When the server receives a get request that includes a Differential-ID field, it can look in its file cache for both versions of the requested file using the content identifiers specified by the Content-ID field and the Differential-ID field. If both version of the file are found, the server can compute the difference between the two files and return the diff rather than the entire file.

When a server replies with a differential update, it must include both the content identifier of the resulting file in the Content-ID field, as well as the content identifier of the file to which the update should be applied in the Differential-ID field.

If the server does not have access to the version of the file that is indicated by the differential content identifier, it can ignore the differential content identifier, and return the entire requested file. Also, if the diff is not smaller, the server can decide to the send the entire file instead.

A client can detect when a differential update is returned by examining the mime type of the reply. After verifying the Content-ID and Differential-ID fields in the reply, the client can apply the diff to its current version of the file to generate the desired version of the file.

To ensure interoperability it is necessary to specify a well known format which can be used to describe the differences between two version of any file. The default diff format is the GDIFF format which is defined in a separate proposal. The client can specify additional acceptable differencing formats using the Accept header field.

A simple server can choose to ignore differential get requests altogether, and simply reply with the entire contents of the requested file.

Note that although the example uses the Content-ID field to specify the desired version of a file, this

field can be omitted in order to obtain a differential update to the most recent version. The server should return a "304 Not Modified" reply if the requested resource has not changed.

Here is a table that lists all the possible combinations and successful replies that can occur when specifying the Content-ID and Differential-ID in the HTTP request:

Request Content ID	Request Differential ID	Action	Reply Content ID	Reply Differential ID
no	no	Return the current version of the file.	yes	no
yes	no	Return the correct version of the file as indicated by the Content-ID.	yes	no
no	yes	Return the diff between the current version and the version identified by the Differential-ID.	yes	yes
yes	yes	Return the diff between the version identified by the Content-ID and the version identified by Differential-ID.	yes	yes

Differential Index Retrieval

It is very important that obtaining the most up-to-date index is an efficient operation because clients will poll for updates repeatedly. It would be wasteful to send the entire index each time an index request is made, especially since there will often be little or no change since the last request. Also note that indexes can contain many files, and as a result they can be rather large.

To avoid downloading the complete index repeatedly, a client can use a differential get request to obtain an index. In response to a differential index request the server can reply with the diff between the client's index and the server's current index.

The server decides what content identifier to use for each index. Each content identifier must uniquely identify the index. One way of doing this would be to construct an identifier out of a checksum computed on the index; another way would be by combining the URL of the index with a version number.

Note that a differencing algorithm can be defined that performs a structural comparison of two indexes, rather than a textual comparison.

2.6 HTTP Proxy Caching

An HTTP proxy can be made aware of the Content-ID and Differential-ID fields in HTTP requests and replies. Because the content identifier is included in each GET request, the proxy can avoid accidentally returning the wrong version of a requested file. Getting the wrong version from an intermediate proxy often presents a serious problem when distributing mission critical applications and data.

The proxy can use the content identifier field to uniquely identify the content that is being transferred. The same piece of content, even when downloaded from multiple locations, is likely to have the same content identifier. The proxy can use this fact to avoid multiple redundant downloads.

In addition, a proxy can use the Differential-ID header field to reply to differential GET requests. If both versions of the file are in the proxy's cache, the proxy can compute the differential reply.

2.7 Backward Compatibility

The DRP protocol works efficiently in existing HTTP enabled environments. Some special precautions need to be taken to make sure that older proxy servers don't interfere with the versioning strategy used by the DRP protocol.

HTTP/1.0 Proxy Caching

To avoid the incorrect caching of data files in HTTP/1.0 proxies, the server should mark the files returned to a HTTP/1.0 client or proxy as not cacheable. This can be done using the Pragma and/or Expires fields, as defined in the HTTP/1.0 specification.

HTTP/1.1 Proxy Caching

To correctly cache files in an HTTP/1.1 proxy, the server can use the Vary field in the HTTP reply which should be set to "Content-ID". Here is an example of a reply which includes a Content-ID:

```
HTTP/1.1 200 OK
Vary: Content-ID
Content-ID: urn:md5:PEFjWBDv/sd9alS9BYuX0w==
...
```

If the request also contains a Differential-ID, the Vary header field should be set to "Content-ID,Differential-ID". Here is an example of a reply to a differential GET request:

```
HTTP/1.1 200 OK
Vary: Content-ID,Differential-ID
Content-ID: urn:md5:PEFjWBDv/sd9alS9BYuX0w==
Differential-ID: urn:md5:3FS2oCnWPZptpN05oKBemA==
```

Using this strategy, it is clear that client requests that specify a Content-ID will be cached appropriately by an HTTP/1.1 proxy. However, when the Content-ID is omitted, the proxy will not match the request and will contact server each time.

Using the Vary header this way provides the best performance over HTTP/1.1 without weakening any of the caching guarantees required by the DRP protocol.

Compatibility with Existing Servers

The DRP protocol is defined so that it does not necessarily require any server support. The index can be specified using a normal file on a server that describes part or all of the site. It is easy to generate an index file using a simple program that scans the file system.

The client can use the index file to determine which files to download from the site using normal HTTP request. If the server does not provide special DRP support, no guarantees can be given about file

versioning. However, if the index file specifies content identifiers for each file, the client can use these to verify that it has obtained the correct version by recomputing the appropriate checksums.

Note that in this manner DRP can also be used with a number of existing URL types such as "ftp:" and "file:".

3. Conclusion

The HTTP Distribution and Replication Protocol provides significant added value to the HTTP protocol. It enables the distribution and replication of data files in a simple and efficient manner. DRP is a widely applicable technology and is appropriate for the distribution of HTML-based content as well as mission critical applications.

The DRP defines the following new features:

- Content identifiers, using the existing URI specification, which can uniquely identify a piece of content.
- An index format which can be used to describe a set of files.
- A new HTTP header field, Content-ID, which is used to obtain the correct version of a file by specifying a content identifier.
- A new HTTP header field, Differential-ID, which is used to obtain a differential update for a file.

3.1 References

- "Naming and Addressing: URLs" http://www.w3.org/Addressing/
- "Uniform Resource Locators (URL)" by T. Berners-Lee, L. Masinter, M. McCahill. http://www.w3.org/Addressing/rfc1738.txt
- "URN Syntax" http://ds.internic.net/rfc/rfc2141.txt
- "Uniform Resource Names (urn)" http://www.ietf.org/html.charters/urn-charter.html
- "The MD5 Message Digest Algorithm" by R. Rivest. http://www.rsa.com/pub/rfc1321.txt
- "Secure Hash Standard", U.S. Department of commerce, National Institute of Standards and Technology. http://csrc.nist.gov/fips/fip180-1.txt
- "MIME (Multipurpose Internet Mail Extensions)" by N. Borenstein, N. Freed http://www.w3.org/Protocols/rfc1341/0_Abstract.html
- "Extensible Markup Language (XML)" by Tim Bray, C. M. Sperberg-McQueen. http://www.w3.org/TR/WD-xml-961114.html
- "Meta Content Framework Using XML" by R.V. Guha, Tim Bray. http://www.w3.org/TR/NOTE-MCF-XML/
- "Hypertext Transfer Protocol -- HTTP/1.0" by T. Berners-Lee, R. Fielding, H. Frystyk. http://www.w3.org/Protocols/rfc1945/rfc1945
- "Hypertext Transfer Protocol -- HTTP/1.1" by R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. http://www.w3.org/Protocols/rfc2068/rfc2068
- "Hypertext Transfer Protocol - Next Generation". http://www.w3.org/Protocols/HTTP-NG/
- "Network Performance Effects of HTTP/1.1, CSS1, and PNG " by Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Wium Lie, Chris Lilley. http://www.w3.org/pub/WWW/Protocols/HTTP/Performance/Pipeline.html
- "Generic Diff Format Specification" by Arthur van Hoff and Jonathan Payne. "http://www.marimba.com/developer/proposals/GDIFF.html
- "HTTP: Delta-Encoding Notes" by Stephen Williams.

http://ei.cs.vt.edu/~williams/DIFF/prelim.html
- "Potential benefits of delta encoding and data compression for HTTP" Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Proceedings SIGCOMM '97, Cannes, France
http://ftp.digital.com/%7emogul/sigcomm97.ps
http://www.research.digital.com/wrl/techreports/abstracts/97.4.html

Appendices

A. Index DTD

Below is the XML Document Type Definition used in this document to describe an index:

```
<!-- XML DTD for the DRP Protocol -->

<!ELEMENT index     (file | dir)*                              >
<!ATTLIST index     id              CDATA       #IMPLIED       >
<!ATTLIST index     base            CDATA       #IMPLIED       >

<!ELEMENT file      EMPTY                                      >
<!ATTLIST file      path            CDATA       #REQUIRED      >
<!ATTLIST file      id              CDATA       #IMPLIED       >
<!ATTLIST file      size            CDATA       #IMPLIED       >
<!ATTLIST file      mime            CDATA       #IMPLIED       >
<!ATTLIST file      info            CDATA       #IMPLIED       >

<!ELEMENT dir       (file | dir)*                              >
<!ATTLIST dir       path            CDATA       #REQUIRED      >
```

Notes:

- A valid index should start with the following XML declaration:

```
<?XML VERSION="1.0" RMD="NONE"?>
```

- The mime type of an index is **application/drp-index**.
- An index describes a hierarchical set of resources. The **dir** element represents a directory and can contain further elements, which describe its content.
- The **id** attribute is a content identifier as defined in this proposal.
- The **base** attribute of the **index** element is used to override the default base URL of the index. The URL can be relative or absolute.
- The **path** attribute of the **file** and **dir** elements is a relative path to the parent directory of the node.

- The **size** attribute of the **file** element is the size of the file in bytes.
- The **mime** attribute of the **file** element specifies the mime type of the file.
- The **info** attribute of the **file** element can be used to specify further attributes of the file such as whether it is executable or not.